Neural Nets and genetic algoritms

Tobias Nielsen - 200975

December 17, 2006

Contents

1 Neural nets

For this first task, i have created a trivial perlscript that aids me in testing the different parameter's fairly easy.

The script has been made as a frontend to the *rbp* program.

The scripts handles a fair treatment of data, training and evaluation data by dividing up the data in tree chunks of data. The test and evaluation data is selected by sliding a window through the full data as visualized in the figure below.

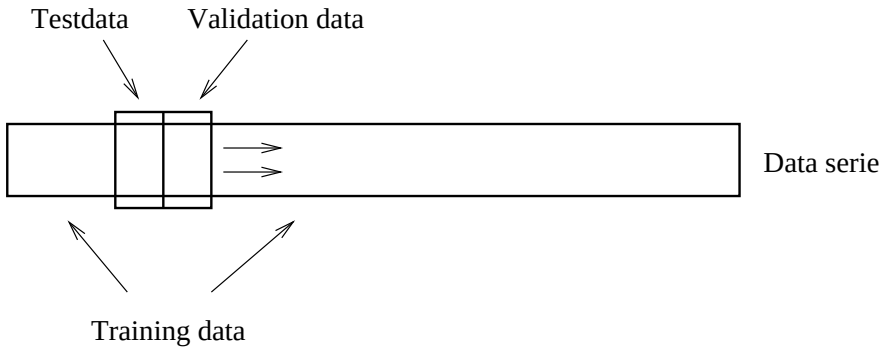

Figure 1: Sliding window in data structure

After this a mean value of the result of the training, testing and evaluation is being calculated and visualized as a metric of the system in focus.

The system is always executedd with following parametres if nothing else is stated (raw copy from perl source)

```
tolerance     => 0.1,
eta           => 0.1,
momentum      => 0.5,
seed          => 7,
weights       => 1,
maxruns       => 4000,
numTrainData  => 5,
numEvalData   => 5
```

Table 1: Default params for execution of the rbp program

Since i keep the focus on the actual performance of the network and not on specific coding, I will leave out the code for the program here since its trivial to implement a simular script in awk, perl or a simular scripting language.

1.1 Sub task A

First i have to perform a clean run with all the default data set.

The first runs gives me an average success rate on the training set of 10% solved. This is offcourse not good.

We then try it with a more aggressive random function with a range of ± 10 which gives almost the same result.

The reason for this is that the net is implemented by using the sigmoid function which is only able to provide an output between 0 and 1. Therefore it is fair to expect that the system will perform better if we scale the output so its in this range.

1.2 Sub task B

By simply rescaling the output data by a form similar to the following formula:

$$NewO_n = O_n \cdot \frac{1}{2 \cdot \max(O)} + 0.5$$

The part here that is interesting is that the output will be forced to be in the whole aspect of the output function. This is not a good solution since we are using the sigmoid function which only approaches one as its input goes towards infinity and zero as it goes towards minus infinity. Therefore its wiser to chose it a bit lower so therefore we simply chose the formula to be like this instead:

$$NewO_n = O_n \cdot \frac{1}{2 \cdot 10} + 0.5$$

After scaling, we perform a test where we vary the η and the α values. The η goes from 0.1 $\longrightarrow$ 0.9 in steps of 0.1 while we also tests the α in four steps: 0.0 (no momentum), 0.1 (a little momentum), 0.5 (mean momentum) and 0.9 (high momentum).

The result is as follows:

```
A: moment, n: eta, NI: number of iterations before best error
MNI: The number of iterations before training converged, Eval: Evaluation test.
A: 0.0 n: 0.1 Avg: 60.000 % 0.09597 NI: 29.3 MNI: 214.0 - Eval: 48.89 % 0.11511
A: 0.1 n: 0.1 Avg: 59.048 % 0.09672 NI: 26.9 MNI: 192.5 - Eval: 48.89 % 0.11507
A: 0.5 n: 0.1 Avg: 59.048 % 0.10050 NI: 23.6 MNI: 108.7 - Eval: 48.89 % 0.11539
A: 0.9 n: 0.1 Avg: 63.810 % 0.09999 NI: 15.8 MNI: 37.2 - Eval: 47.78 % 0.13317

A: 0.0 n: 0.2 Avg: 53.333 % 0.12006 NI: 119.1 MNI: 651.5 - Eval: 39.84 % 0.15526
A: 0.1 n: 0.2 Avg: 49.524 % 0.12017 NI: 51.6 MNI: 289.5 - Eval: 42.22 % 0.14538
A: 0.5 n: 0.2 Avg: 60.000 % 0.10234 NI: 15.1 MNI: 52.4 - Eval: 50.48 % 0.11383
```

```
A: 0.9 n: 0.2 Avg: 60.000 % 0.10721 NI: 14.9 MNI: 26.7 - Eval: 43.65 % 0.15353

A: 0.0 n: 0.3 Avg: 58.095 % 0.12423 NI: 547.0 MNI: 2286.5 - Eval: 29.52 % 0.20171
A: 0.1 n: 0.3 Avg: 57.143 % 0.11233 NI: 343.5 MNI: 1448.6 - Eval: 37.46 % 0.17482
A: 0.5 n: 0.3 Avg: 52.381 % 0.12531 NI: 45.5 MNI: 141.3 - Eval: 43.02 % 0.14108
A: 0.9 n: 0.3 Avg: 60.000 % 0.11654 NI: 63.7 MNI: 298.9 - Eval: 43.65 % 0.14840

A: 0.0 n: 0.4 Avg: 49.524 % 0.16874 NI: 573.8 MNI: 4000.0 - Eval: 19.84 % 0.27521
A: 0.1 n: 0.4 Avg: 33.333 % 0.17286 NI: 1028.5 MNI: 3562.9 - Eval: 24.60 % 0.23646
A: 0.5 n: 0.4 Avg: 49.524 % 0.14941 NI: 467.1 MNI: 1073.7 - Eval: 38.89 % 0.18638
A: 0.9 n: 0.4 Avg: 64.762 % 0.12674 NI: 651.9 MNI: 2849.7 - Eval: 32.06 % 0.23907

A: 0.0 n: 0.5 Avg: 48.571 % 0.16248 NI: 732.9 MNI: 4000.0 - Eval: 7.94 % 0.36500
A: 0.1 n: 0.5 Avg: 43.810 % 0.16937 NI: 818.1 MNI: 4000.0 - Eval: 8.73 % 0.33956
A: 0.5 n: 0.5 Avg: 55.238 % 0.13089 NI: 343.8 MNI: 2839.8 - Eval: 34.60 % 0.20447
A: 0.9 n: 0.5 Avg: 65.714 % 0.15292 NI: 1076.2 MNI: 3270.5 - Eval: 6.35 % 0.36429

A: 0.0 n: 0.6 Avg: 37.143 % 0.18689 NI: 572.9 MNI: 4000.0 - Eval: 5.56 % 0.42387
A: 0.1 n: 0.6 Avg: 43.810 % 0.15470 NI: 947.1 MNI: 4000.0 - Eval: 9.52 % 0.38888
A: 0.5 n: 0.6 Avg: 59.048 % 0.14748 NI: 1027.1 MNI: 3841.0 - Eval: 17.46 % 0.28309
A: 0.9 n: 0.6 Avg: 62.857 % 0.11036 NI: 1589.5 MNI: 3748.9 - Eval: 9.68 % 0.35795

A: 0.0 n: 0.7 Avg: 47.619 % 0.15134 NI: 893.8 MNI: 4000.0 - Eval: 5.72 % 0.37993
A: 0.1 n: 0.7 Avg: 50.476 % 0.14198 NI: 1204.8 MNI: 4000.0 - Eval: 8.25 % 0.37717
A: 0.5 n: 0.7 Avg: 53.333 % 0.14647 NI: 720.5 MNI: 3938.5 - Eval: 13.02 % 0.31374
A: 0.9 n: 0.7 Avg: 57.143 % 0.13547 NI: 1276.2 MNI: 4000.0 - Eval: 6.35 % 0.39775

A: 0.0 n: 0.8 Avg: 39.048 % 0.15658 NI: 793.3 MNI: 4000.0 - Eval: 7.14 % 0.41579
A: 0.1 n: 0.8 Avg: 38.095 % 0.16927 NI: 1138.1 MNI: 4000.0 - Eval: 7.94 % 0.36448
A: 0.5 n: 0.8 Avg: 57.143 % 0.18183 NI: 1345.7 MNI: 4000.0 - Eval: 11.91 % 0.34615
A: 0.9 n: 0.8 Avg: 53.333 % 0.15273 NI: 1264.3 MNI: 4000.0 - Eval: 5.56 % 0.41922

A: 0.0 n: 0.9 Avg: 33.333 % 0.18981 NI: 827.6 MNI: 4000.0 - Eval: 5.56 % 0.42584
A: 0.1 n: 0.9 Avg: 34.286 % 0.20470 NI: 1216.2 MNI: 4000.0 - Eval: 6.35 % 0.41298
A: 0.5 n: 0.9 Avg: 62.857 % 0.13415 NI: 1838.6 MNI: 4000.0 - Eval: 5.56 % 0.39886
A: 0.9 n: 0.9 Avg: 54.286 % 0.15042 NI: 1880.5 MNI: 4000.0 - Eval: 3.18 % 0.44752
```

It is quickly seen that the network trains well with a small η and a medium α in the system. It seems that a quick scan shows that the test sets reach a maximum with an η at 0.1 and a α at 0.9. The validation shows a α of 0 and an η of 0.1 is the best result. But a look at the fastest convergence shows that it is with an η of 0.2 and a α of 0.9

When the η gets larger, it starts to take longer time for the system to learn when we apply a α to the system. The reason for this is presumeable because the system tends to be erratic and therefore the α cuts the performence down instead of improving it.

1.3 Sub task C

By looking at the inverse conversion function which will be like follows:

$$O_{real} = (O_{result} - 0.5) \cdot 2 \cdot 10$$

We can easy conclude from above that a tolerance can move the real output 2 steps in either direction of the correct answer.

If we pick a single run with an η of 0.1 and a α of 0.1, the output becomes:

```
T: 0.10 Avg: 63.810 % 0.09877 NI: 250.4 MNI:  846.0 - Eval: 47.78 % 0.15291
T: 0.09 Avg: 62.857 % 0.09617 NI: 264.2 MNI: 1071.6 - Eval: 39.84 % 0.15985
T: 0.08 Avg: 60.000 % 0.09242 NI: 459.4 MNI: 1758.6 - Eval: 33.49 % 0.16983
T: 0.07 Avg: 59.048 % 0.09039 NI: 537.4 MNI: 3390.0 - Eval: 28.73 % 0.18049
T: 0.06 Avg: 54.286 % 0.08957 NI: 556.3 MNI: 3647.0 - Eval: 27.14 % 0.17734
T: 0.05 Avg: 48.571 % 0.08830 NI: 582.1 MNI: 3811.1 - Eval: 23.97 % 0.17743
```

It is certain that the smaller tolerance forces the system to train harder in order to reach a goal and therefore the efficency drops slowly as its not able to reach a goal as easy as before. But another thing is happening - the system is training further before it reach a local minuma and thereby reaching a smaller error. I suppose the reason for this is that the network is not trained on celldata that is within the tolerance - thereby being more fit.

1.4 Sub task D

By changing the seed the random number generater is started in a different position than before and thefore gives the system completly new possibilities. This can give a better result and it can also provide a completely unsolveable result that forces the weights in to a local minima by which it can get out of. I have here picked some results:

```
S: seed number
S:  7 Avg: 59.048 % 0.09672 NI: 26.9 MNI: 192.5 - Eval: 48.89 % 0.11507
S: 10 Avg: 47.619 % 0.13118 NI: 42.3 MNI: 417.9 - Eval: 38.25 % 0.16395
S: 15 Avg: 66.667 % 0.11464 NI: 34.8 MNI: 306.1 - Eval: 34.60 % 0.14415
S: 20 Avg: 58.095 % 0.12115 NI: 78.4 MNI: 846.3 - Eval: 43.02 % 0.15955
S: 25 Avg: 45.714 % 0.13183 NI: 44.7 MNI: 266.6 - Eval: 46.98 % 0.14585
```

As can be seen, a seed of 15 gives a good result for the test data, but not for the evaluation data - a seed of 25 gives better results for the evaluation data.

1.5 Sub task E

Here i will perform a few test to see how the system varies with the number of internal neurons

```
in: Number of inner neurons
in 7  Avg: 59.048 % 0.09672 NI:  26.9 MNI: 192.5 - Eval: 48.89 % 0.11507
in 8  Avg: 62.857 % 0.09069 NI:  27.0 MNI: 408.2 - Eval: 49.05 % 0.12159
in 9  Avg: 59.048 % 0.09932 NI:  32.0 MNI: 280.6 - Eval: 47.46 % 0.11803
in 6  Avg: 61.905 % 0.10232 NI:  50.8 MNI: 490.4 - Eval: 51.90 % 0.11852
in 5  Avg: 60.000 % 0.11110 NI:  90.2 MNI: 778.0 - Eval: 36.67 % 0.15025
in 4  Avg: 60.952 % 0.10410 NI:  84.4 MNI: 493.0 - Eval: 39.84 % 0.14984
in 3  Avg: 69.524 % 0.09571 NI:  83.6 MNI:1018.1 - Eval: 47.14 % 0.12184
in 2  Avg: 65.714 % 0.10647 NI: 228.5 MNI:2214.1 - Eval: 47.46 % 0.13018
in 7x Avg: 46.667 % 0.16674 NI: 905.2 MNI:4000.0 - Eval: 12.70 % 0.35603
```

As it can be seen there is little difference between running with 7, 9 or 6 neurons, since it gives a fair output compared to what has been achieved prievious. But reducing it as far as 2 inner neurons shows that the system cannot be trained easily. On the other hand, looking at the error gives an idea that the best result is about 7 or 8 neurons. A last attempt was to connect all inputs directly to the output of the system - This didn't give a good result which indicates that the output isn't directly linearly dependent on the inputs.

2 Genetic algorithms

In this task, i will examine Genetic Algorithms and its abillity to perform with different basic parameters. The program that i will use for this task is the PGA program by Peter Ross

2.1 Experiment 1

In the first experiment i will first test what influence the chromosome length has on the outcome. I start out with the default length of 32 bits and the max-1 problem. The Max-1 will be used for the rest of this experiment I have initially set the migration and the mutation rate to 0.

The result as can be seen from one experimental run is as follows:

```
Generation: 730      Evaluations so far: 3900
   Pop.......Average..........Best.(max = 32)
   0   =      32.0000000       32.0000000
   1   =      32.0000000       32.0000000
   2   =      30.0000000       30.0000000
   3   =      30.0000000       30.0000000
   4   =      30.0000000       30.0000000
```

In this run, its only two of the five populations which reach a complete result. The last tree is missing two bits each.

I repeat the task five times and gets simular results.

This is a risk when all the chromosomes from a single population has a specific bit not set in either of the chromosomes. This will newer be altered since the population is newer allowed to mutate - only crossover. An crossover with an external population can aid in helping this problem - but that is outside the scope of this experiemnt.

As i increase the chromosome length, the situation seems to worsen even more as can be seen below with a length of 50 and 100:

```
    Generation: 780      Evaluations so far: 4150
Pop.......Average..........Best.(max = 50)
0   =      46.0000000       46.0000000
1   =      49.0000000       49.0000000
2   =      49.0000000       49.0000000
3   =      46.0000000       46.0000000
4   =      46.0000000       46.0000000
```

```
    Generation: 3540     Evaluations so far: 17950
Pop.......Average.........Best.(max = 100)
0   =      81.0000000       81.0000000
1   =      82.0000000       82.0000000
2   =      82.0000000       82.0000000
3   =      85.0000000       85.0000000
4   =      85.0000000       85.0000000
```

By increasing the population size, the tendency can be lowered somewhat as can bee seen by the below run - we show here with a chromosome length of 100 and population size of 100 and 200.

```
    Generation: 2230     Evaluations so far: 11650
Pop.......Average.........Best.(max = 100)
0   =      91.0000000       91.0000000
1   =      91.0000000       91.0000000
2   =      92.0000000       92.0000000
3   =      89.0000000       89.0000000
4   =      90.0000000       90.0000000

    Generation: 4360     Evaluations so far: 22800
Pop.......Average.........Best.(max = 100)
0   =      99.0000000       99.0000000
1   =     100.0000000      100.0000000
2   =      97.0000000       97.0000000
3   =      99.0000000       99.0000000
4   =      98.0000000       98.0000000
```

So this shows that the system can actually converge with a large enough population. By having a larger population, the chance that all bits set spread over a single population is bigger.

Generally it is quite likely that all the different bits will be represented in a smaller population, but since some of the chromosomes sometimes is not selected for breeding, vital bits can also be lost in this way.

The number of generations also increases since there is now wider range of possible breed solutions available.

If i try to increase the population we can also raise the statistical chance that an optimal solution will be found. It is therefore raised to 400:

```
    Generation: 8430     Evaluations so far: 44150
```

```
Pop.......Average.........Best.(max = 100)
0   =     100.0000000      100.0000000
1   =     100.0000000      100.0000000
2   =     100.0000000      100.0000000
3   =     100.0000000      100.0000000
4   =     100.0000000      100.0000000
```

Another possible choice could be to allow crossover to produce twins. This reduces the chance for loss of bits and therefore could convert faster.

The repetition of the above experiment produces the following result:

```
    Generation: 4370     Evaluations so far: 45700
Pop.......Average.........Best.(max = 100)
0   =     100.0000000      100.0000000
1   =     100.0000000      100.0000000
2   =     100.0000000      100.0000000
3   =     100.0000000      100.0000000
4   =     100.0000000      100.0000000
```

This causes the number of generations to drop to nearly half, but off course the number of evaluations happens at almost the same amount

2.1.1 Conclusion

The fact that a single population can tend to converge to a solution in a fixed area can be a problem in a single population. This seems to be possible to alter this by allowing crossover from other islands. This is due to a single island becomming to simular, but compared with another island it may be wery different and therefore it can produce a rather fresh input to a population that is "stuck!!"

2.2 Experiment 2

In this second experiment, i will examine whatever the rank selection method or the fittest selection method proves the best result. First to summarise, i will give a small description of the difference of the two methods.

The fitness selection choose by examining the result of the evaluation of the chromosome. If the chromosomes in question all performs almost similar, they all get an similar chance to breed again on the other hand the better an item does the bigger the amount of breeding's it can produce - even to the point where its the only one breeding. This can be perceived as a way of cake slicing where the better you do, the greater slice you get.

When using rank selection, a forced choice is being made - even if the chromosomes all performs almost similar, the best is always selected even though it may only be slightly better than the rest. Since a selection is enforced, this method is often considered more brutal than the fitness function.

By looking at these initial two runs, its quickly visible that the rank function reaches its goal as do fitness. But it also finds the result faster. So i will choose to use the number of iterations to measure performance over a mean of five or more runs - which gives a macroscopic view of the different runs.

2.3 Experiment 3

Now i will try to examine what kind of aid we get by crossover. The opposite of crossover is only mutation. The first test i will try is by the De Jong 3 (now DJ3) function. This will be tested by first having no crossover and next normal two point crossover plus finally uniform crossover. The final runs with DJ5 and modified binary 6 (now BF6), will be done only with no and two-point crossover.

All runs are performed with initially following arguments:

```
pga -P1 -p100 -srank
```

Then the result is presented in table

Eval	crossover	mean iterations
DJ3	none	2000 - 3000
DJ3	two	1300 - 1500
DJ3	uniform	1200 - 1600
DJ5	none	2600 - 7100
DJ5	two	2500 - 4600
BF6	none	2000 - 3000
BF6	two	2500 - 3500
BF6	uniform	4000 - 9000

Table 2: Running with different types of crossover

It seems in table 2, there are some differences in the matter of the evaluation problem since it clear that the last BF6 evaluation function doesn't perform two well in the situation. I suppose for this situation that the reason the crossover performs so bad as it does in these scenarios is that the crossover is allowed to switch in all positions.

It again in table 3 on the next page, depends on the problem what ever the results become. It seems for the DJ3 problem seems to suffer when the mutation rate was changed from it orriginal 1 bit per cromosome to a tree bit mutation. The BF6 problem tends to improwe slightly when the amounts of bits changed per mutation increases.

Eval	mutation	mean iterations
DJ3	0.03125	1300 - 1500
DJ3	0.062	2800 - 3500
DJ3	0.10	2800 - 3500
DJ5	0.03125	2600 - 4000
DJ5	0.062	3000 - 4000
DJ5	0.1	3000 - 4000
BF6	0.03125	3000 - 5000
BF6	0.062	3000 - 5000
BF6	0.10	2800 - 4000

Table 3: Running with different quantities of mutation (1,2,3 bits)

2.3.1 Conclusion

In this final experiment i had a single idea that i considered severel times, but could not see how i could get the PGA program to perfrom: By having crossover in a direct binary form, its quite likely that a single bit mutation can cause a huge difference. This is due to dataseries may implementing as several longer than one bit values. That means that a sudden crossover in the middle of such a value, will have a huge a huge impact and therefore move it a far way out of its current area. This can be avoided by using some level of scheming to avoid violent crossovers - but i could not see how i could get PGA to do such a task.